AD
Reports Control Symbol
OSD-1366

Research and Development Technical Report
ECOM-01901-28

AD 661309

GRAPHICAL-DATA-PROCESSING RESEARCH STUDY AND EXPERIMENTAL INVESTIGATION

SIXTH QUARTERLY REPORT

By

P. E. Hart J. H. Munson

SEPTEMBER 1967

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

ECOM
UNITED STATES ARMY ELECTRONICS COMMAND · FORT MONMOUTH, N.J. 07703
CONTRACT DA 28-043 AMC-01901(E)
STANFORD RESEARCH INSTITUTE
Menlo Park, California 94025

40

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

The citation of trade names and names of manufacturers in this report is not to be construed as official Government endorsement or approval of commercial products or services referenced herein.

Disposition

Destroy this report when it is no longer needed. Do not return it to the originator.

TECHNICAL REPORT ECOM-01901-28 SEPTEMBER 1967

GRAPHICAL-DATA-PROCESSING RESEARCH STUDY AND EXPERIMENTAL INVESTIGATION

REPORT NO. 28, SIXTH QUARTERLY REPORT

1 JUNE TO 31 AUGUST 1967

SRI Project 5864

CONTRACT DA 28-043 AMC-01901(E)

Continuation of Contract DA 36-039 AMC-03247(E)

Prepared By

P. E. HART J. H. MUNSON

STANFORD RESEARCH INSTITUTE

MENLO PARK, CALIFORNIA 94025

For

U.S. ARMY ELECTRONICS COMMAND, FORT MONMOUTH, N.J. 07703

ABSTRACT

This report describes the continuing development of scanning, preprocessing, character-classification, and context-analysis techniques for hand-printed text, such as computer coding sheets in the FORTRAN language.

The performance of topological feature extraction, combined with character classification by a learning machine, is described, and compared with the performance of other combinations. By performing intra-author testing (gathering the training data and test data from the same author), we have achieved a dramatic reduction in test error rate, to less than 10 percent on a limited sample.

We describe an experiment in which a fragment of FORTRAN text is scanned, preprocessed, classified character by character, and subjected to context analysis to greatly reduce the recognition-error rate. Finally, we discuss advances in the method of analyzing arithmetic expressions, a key aspect of the context analysis.

CONTENTS

ILLUSTRATIONS

TABLE

I INTRODUCTION

This report describes the continuing development of scanning, preprocessing, character-classification, and context-analysis techniques for hand-printed text. The particular subject matter of our investigation is hand-printed FORTRAN text on standard computer coding sheets, with a 46-character alphabet. The reader is referred to the previous reports of this project for background and supplementary material.

In Sec. II, we describe experiments in which character images are preprocessed by the topological feature extraction program TOPO 2 and classified by various methods. Comparisons are made of the performance of different classifiers and different preprocessors.

The effect of intra-author testing, in which the training and testing data are taken from the same individual author, is reported in Sec. III. On a limited sample of test alphabets, intra-author testing produced a dramatic decrease in recognition error rate, from approximately 20 percent to less than 10 percent.

In Sec. IV, we follow a fragment of FORTRAN text from the coding sheet through scanning, preprocessing, classification, and context analysis. In this example, statement-by-statement syntax analysis reduced the per-character error rate by a factor of ten. This description provides a good example of the methods we are developing to improve text recognition through context analysis.

We conclude with a discussion of the problem of analyzing algebraic expressions, which is presently the most important problem area of the syntax analysis. A search through all possible alternative character strings to find a legal expression is prohibitive in terms of both memory and time required. We describe several methods that contribute to reducing the search to manageable proportions.

II LEARNING-MACHINE AND NEAREST-NEIGHBOR EXPERIMENTS ON THE TOPOLOGICAL FEATURES FROM TOPO 2

A. Introduction

In the Fifth Quarterly Report we described TOPO 2, a program for the SDS 910 computer that extracted topological and geometrical features from hand-printed character images and classified the characters on the basis of these features. We mentioned that we planned experiments in which the features generated by TOPO 2 would be used as the basis for classification of the characters by a learning machine. This section describes some of these experiments. Additional experiments, in which the training and testing pattern sets were derived from the same author, are reported in the next section.

The TOPO 2 program was modified so that, as it processed each character image, it produced an output feature vector containing 64 topological feature coordinates. Sixteen of these features were related to the concavities of the figure, sixteen were related to the spurs, and ten described the enclosures, height, and width. The remaining 22 features resulted from the special calculations performed in TOPO 2 to discriminate among groups of similar categories, such as 5 and S. The file of feature vectors was used for training and testing the CALM learning-machine simulation, in the same manner that other files of feature vectors from different processing systems have been used in the past.

The experimental results lead to two types of comparison. On one hand, the performance of CALM on the features from TOPO 2 may be compared with the performance of the classification routines that were hand-coded into TOPO 2 itself, since both classifiers were working with the same feature vectors. On the other hand, the success of the topological feature extraction may be evaluated through comparison with the performance of CALM working on feature vectors from other preprocessors.

B. TOPO-CALM Experiment 3

The training set for TOPO-CALM Experiment 3 contained 56 FORTRAN alphabets, or 2576 characters. Forty-nine of the training alphabets were obtained by taking the first alphabet of each of the authors in our standard data base. The author of the test characters was represented by one of these 49 training alphabets, and by seven additional alphabets scattered among the 56 in the training set. Thus, the training set may be considered to have an intra-author enrichment of 8/56, or 14 percent. The test results were probably somewhat better than would be obtained in a completely independent-author (inter-author) test. The effect of intra-author testing is dealt with more fully in the next section.

The test set consisted of two alphabets (92 characters). A 46-category linear learning machine was used, with the training margin set to 200 (approximating the average squared length of a pattern vector). After the second iteration, the training margin was reduced to 50, in the hope of allowing the training activity to settle down. Whether or not this occurred to a significant degree is not clear from the results.

The learning curves are shown in Fig. 1. The training error rate reached 19 percent in five iterations; the test error rate reached 19 percent, although it rose to 22 percent in the last iteration. It may be noted that the training and test error rates reached approximately the same values. This closing of the usual test-training error-rate gap reflects both the large size of the training set (the second largest used in any experiment to date) and the enrichment of the training set by patterns from the test author.

C. TOPO-Nearest Neighbor Experiment 1

The training and testing sets of pattern-feature vectors that were used in TOPO-CALM Experiment 3 were also used for classification by a type of classifier, new to this project, usually called a nearest neighbor (NN) classifier. The NN classifier shares with the learning machines the property that the classification of test patterns is

LEARNING CURVES
-------- ------

SRI PROJECT 5864, EXPT NO. TOPO-CALM 3
TOPO-CALM EXPT. WITH ENRICHED 56-ALPHABET TRAINING SET.
THERE ARE 2576 TRAINING PATTERNS AND 92 TESTING PATTERNS

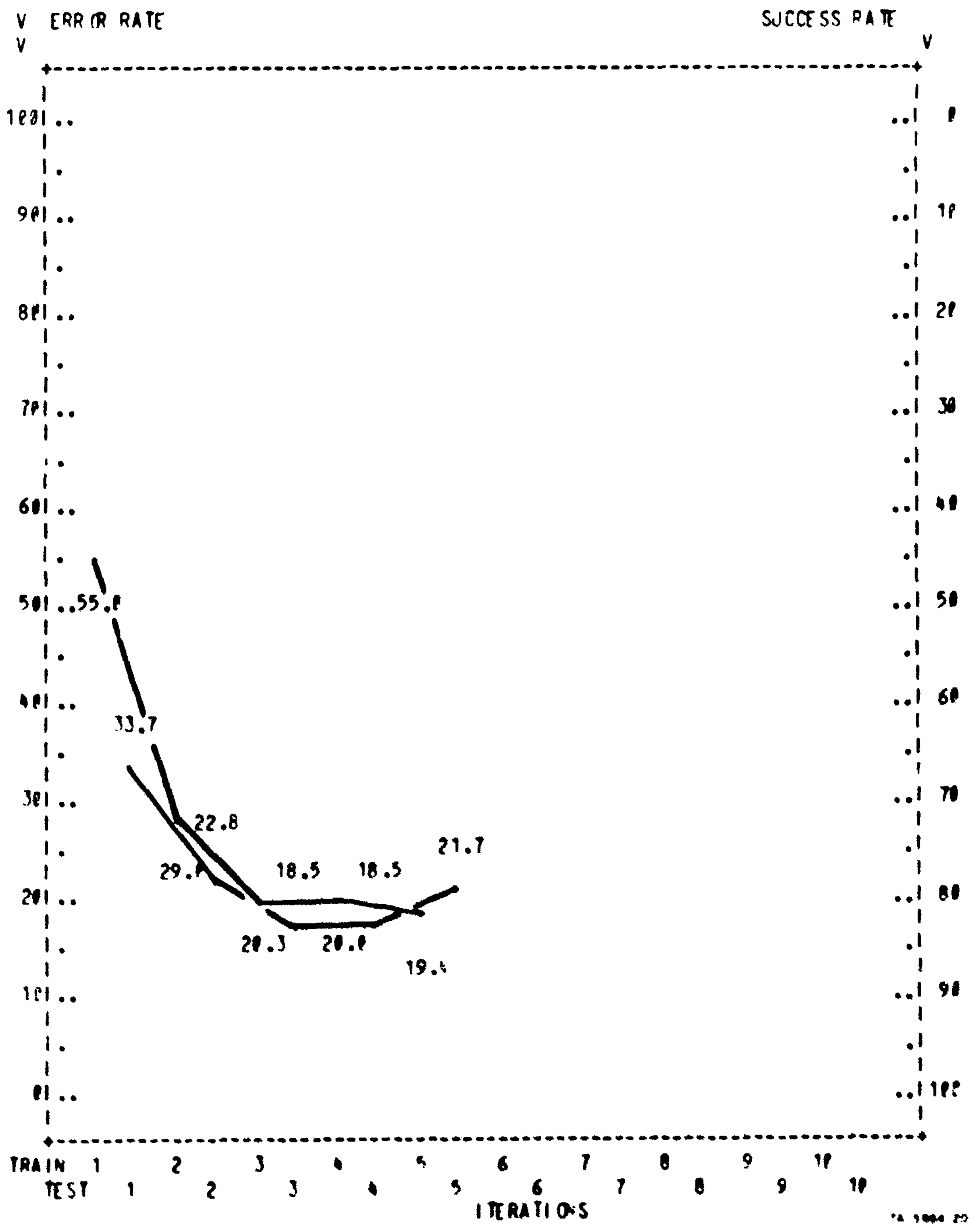


FIG 1 LEARNING CURVES FOR TOPO-CALM EXPERIMENT 3

based on information derived from a set of training patterns. In a learning machine, the information is represented by an array of adaptive weights developed during iterative presentation of the training patterns. In the NN classifier, however, the training patterns themselves are stored as reference vectors against which a test pattern is compared.

In a simple NN machine, all the training (or reference) patterns are stored. The test pattern is compared against each training pattern, and is assigned the category associated with the training pattern closest to it. The measure of closeness, in the classifier we used, is the sum of the absolute differences between the training pattern and the test pattern in each coordinate.

Certain variations of the NN classifier have been studied by various researchers, including Dr. Peter Hart of our laboratory and Dr. Tom Cover of Stanford University, and theoretical results have been obtained. In the n-nearest neighbor machine, classification is based on the n training patterns nearest the test pattern. We shall present results for a 3-NN machine as well as for a simple NN machine. In the condensed nearest neighbor machine, an attempt is made to conserve storage space by eliminating training patterns that are near other training patterns in the same category. The condensed NN machine was not used in our experiment.

In TOPO-NN Experiment 1, we used exactly the same 2576 training patterns (feature vectors) and 92 test patterns as in TOPO-CALM Experiment 3. We simultaneously implemented a simple NN classifier and a 3-NN classifier. For both classifiers, the error rate was 19/92, or 21 percent; in fact, the errors made by the two classifiers were largely the same.

D. Comparison of Results

We are now in a position to compare the performance of four different classifiers in attempting to classify the same test set of 92 feature vectors from TOPO 2. The four classifiers were: the NN machine, the 3-NN machine, the linear learning machine (CALM), and the classification routines of TOPO 2 itself. The first three of these classifiers used the same training set of 2576 characters. For TOPO 2, a training set of sorts was implicit in the 20 alphabets of characters examined by the human experimenter while he developed the classification routines.

The comparison of performance, on a pattern-by-pattern basis, is presented in Table I. With one exception--the classification of Alphabet No. 59 by TOPO 2--the error rate was always close to 10 errors in an alphabet of 46 patterns. The four classification errors that are circled are due to a programming fault that affected the feature-vector output of TOPO 2. Had this fault been corrected, it may be assumed that these classifications would have been correct. The same statement probably applies to the four errors encircled by the dashed line. In summary, we may say that the results point to a generally uniform error rate close to 20 percent.

Let us now compare these results, taken together, with previous results described in Quarterly Reports 2 through 5. In PREP-CALM Experiment 3, a test error rate of 20-23 percent was achieved with a training set of 36 alphabets from 12 authors, using feature vectors produced by the nine-view simulation of the edge-detecting optical preprocessor (PREP 24A). In PREP-CALM Experiment 9, with an expanded training set of 60 alphabets from 20 authors, this error rate was lowered to 19 percent. In MASK-CALM Experiments 2-4, test error rates of 23-27 percent were obtained using combined feature vectors from the CALMMASK preprocessor and Clemens' technique. Throughout the TOPO 2 experiments (Authors 1-49), rates from approximately 20 percent to 30 percent were observed, with the lowest rates generally occurring on alphabets that were examined during the development of the classification routines.

E. Discussion

We may draw certain general conclusions from the information just presented. First, it is clear that the combination of topological feature extraction (TOPO 2) and classification by learning machine (CALM) was successful. The performance achieved by this combination matched that achieved by the nine-view edge-detecting preprocessor

Table I

COMBINED ERROR MAPS FROM FOUR CLASSIFIERS

Category	Alphabet No. 58				Alphabet No. 59			
	TOPO 2	CALM	NN	3-NN	TOPO 2	CALM	NN	3-NN
1								
2								
3							]	]
4					8	B		6
5					S		I	I
6								
7					9			
8		9		9	2	+	3	3
9								
Ø					B	B		
A								
B	8				Ø	8	6	6
C	6				[			
D			O	O				
E								
F	E							
G								
H	M	N						
I								
J		I						
K								
L								
M								
N						W		
O								
P								
Q	O	O	D	D	P	U	U	U
R					P			
S	5				5			
T								
U								
V								
W					U			
X			*				*	
Y								
Z	2	9	2	2				
[					6			
=								
*								
/	1	1	1	1	1	1	1	1
+	*	.	*	*	*			
-		⊙	*	*		⊙		
.			*	*			⊖	⊖
,	]	]						
$	2	=	2	Z	Ø			
]		7	7	7		7	7	7
Total	11	11	10	10	15	9	9	9

(PREP 24A) and CALM. Further evidence supporting these statements will be presented in the next section.

Second, the nearly identical performance of the four classifiers on the feature vectors from TOPO 2 is evidence to support the view, which we hold, that the exact form of the classifier is probably not crucial to the performance of the system. The following argument may be made. A given set of training (and testing) pattern vectors is equivalent to a collection of points in an N-dimensional feature space. Any classifier, which is developed on the basis of the training patterns, attempts to organize this N-space into regions each containing pattern points from a single category. In every case, the property of spatial proximity in the N-space will underlie the process of breaking up the total volume. This fact is most clearly demonstrated by the nearest-neighbor classifier, but it is also present in the linear learning machine and the TOPO 2 classifier, in which each category acquires a single, convex region of N-space, bounded by, at most, k - 1 hyperplanes, where k is the number of categories. Only if the pattern categories are multimodal or inter-twined in N-space will the performance of a simple classifier be bettered by a classifier capable of more complex decision surfaces, such as a NN machine or a Piecewise Linear learning machine. Furthermore, in such a case, the training set must be extensive enough to define the pattern regions. It is the goal of preprocessing techniques to condense and simplify the pattern regions in N-space--in the ideal limit, to reduce each category to a single point. We thus arrive at the viewpoint that improvements in the preprocessing and changes in the choice of the data sets are more likely to produce significant changes in performance than are refinements in the classifier.

Finally, it can be seen that the many experiments reported to date during the project have involved three separate preprocessors (PREP 24, CALMMASK+Clemens' technique, and TOPO 2), together with a variety of experimental conditions, training sets, test sets, etc., and yet in each case we seem to have arrived at a kind of "20 percent barrier" in test error rate. The question arises, "Is there something inherent in a set of hand-printed characters collected from different authors (approximately 50) that limits, or tends to limit, the performance of a preprocessor-plus-classifier combination?" The use of the phrase "20 percent barrier" is not intended to mean that we propose the existence of a specific barrier to progress, or that 20 percent is a number of particular significance. We must conclude, however, that the fact that various approaches have all arrived at nearly the same level of performance seems suggestive. The question is completely open at present.

A vital aspect, common to all the experiments that we have reported thus far, has been the use of training data from a large number of authors. We have avoided specializing the training data (and, therefore, the trained classifier) to an individual author, because we preferred to develop, if possible, a universal classifier. Recently, however, in an attempt to cross the "20 percent barrier," we tried experiments in which the printing of the same person was used for training and testing (intra-author experiments). The dramatic improvement in performance that resulted from intra-author testing is described in the next section.

III INTRA-AUTHOR EXPERIMENTS

A. Introduction

The variations in form among different characters printed by one author are much less than the variations among characters printed by a large population of authors. Accordingly, intra-author character recognition should present a simpler problem than inter-author recognition, and we could expect a learning-machine system to achieve significantly better performance on an intra-author problem.

The price paid for intra-author training and testing is twofold. First, enough data must be collected for each author to provide an ample training set. Second, the trained machine is specific to one author. In an installation where several authors were using a text-recognition system, each would require his own set of trained weights. These requirements might be met quite readily in an operating system. For example, a user of the system might sign on the first day with a sample of his printing that would be used to generate a set of trained weights for him, to be kept in a peripheral memory and recalled as needed. As the author used the system, further training might be performed on his continuing inputs, to refine and update the set of weights. The questions of how much training to do, and how to identify the true categories of the training characters without interfering with the author's use of the system, are interesting system questions that are beyond the scope of the present discussion.

The authors chosen for these first intra-author experiments were the only two authors for whom 20 hand-printed alphabets had been scanned and quantized, namely, John Munson (JM) and Richard Duda (ROD) of the Applied Physics Laboratory. In addition, some actual FORTRAN coding sheets prepared by these authors in the course of program writing were available. Future intra-author experimentation will require additional data-gathering to cover more authors; hopefully, this data-gathering will also enable us to build up a data base from actual coding sheets.

B. Intra-Author TOPO-CALM Experiments

1. TOPO-CALM Experiment 2

In TOPO-CALM Experiment 2, eight alphabets by JM (Sequence Nos. 50-55 and 58-59) were preprocessed by TOPO 2 and used for training, and two alphabets (Nos. 56 and 57) were used for testing. The learning curves are presented in Fig. 2. The training error rate reached 20 percent in four iterations; the test error rate reached 23 percent. It is noteworthy that in spite of the relatively small training set (368 characters), the difference between training and test error rates was small, and the test error rate was close to the best performance obtained in previous inter-author experiments. These facts were strong evidence for reduced variability among patterns from a single author, and we hurried to perform more extensive experiments.

2. TOPO-CALM Experiment 4

Ten additional alphabets by JM (Sequence Nos. 60-69) were preprocessed by TOPO 2 and added to the existing ten. In TOPO-CALM Experiment 4, alphabets 50 and 51 were used for testing, and the other 18 alphabets comprised the training set. Thus, the size of the training set was more than twice that of Experiment 2.

The learning curves for Experiment 4 are shown in Fig. 3. The training and test error rates fluctuated, but in general reached values on the order of 10 percent. This test error rate is approximately half as large as the best obtained previously, in spite of the modest size of the training set. The result dramatically shows the difference between inter-author and intra-author testing.

3. TOPO-CALM Experiment 6

To provide a further check on the results of Experiment 4, we performed an experiment under identical conditions with characters printed by a different author (ROD). Alphabets No. 20 and No. 21 were used for testing, and the remaining eighteen for training. The results are shown in Fig. 4. The results are slightly better than those of Experiment 4. The training and test error rates are below 10 percent. This result confirms that of Experiment 4.

LEARNING CURVES

SRI PROJECT 5864, EXPT NO. TOPO-CALM 2
INTRA-AUTHOR EXPT. TRAIN ON 8 ALPHABETS, TEST ON NOS. 56,57.
THERE ARE 368 TRAINING PATTERNS AND 92 TESTING PATTERNS

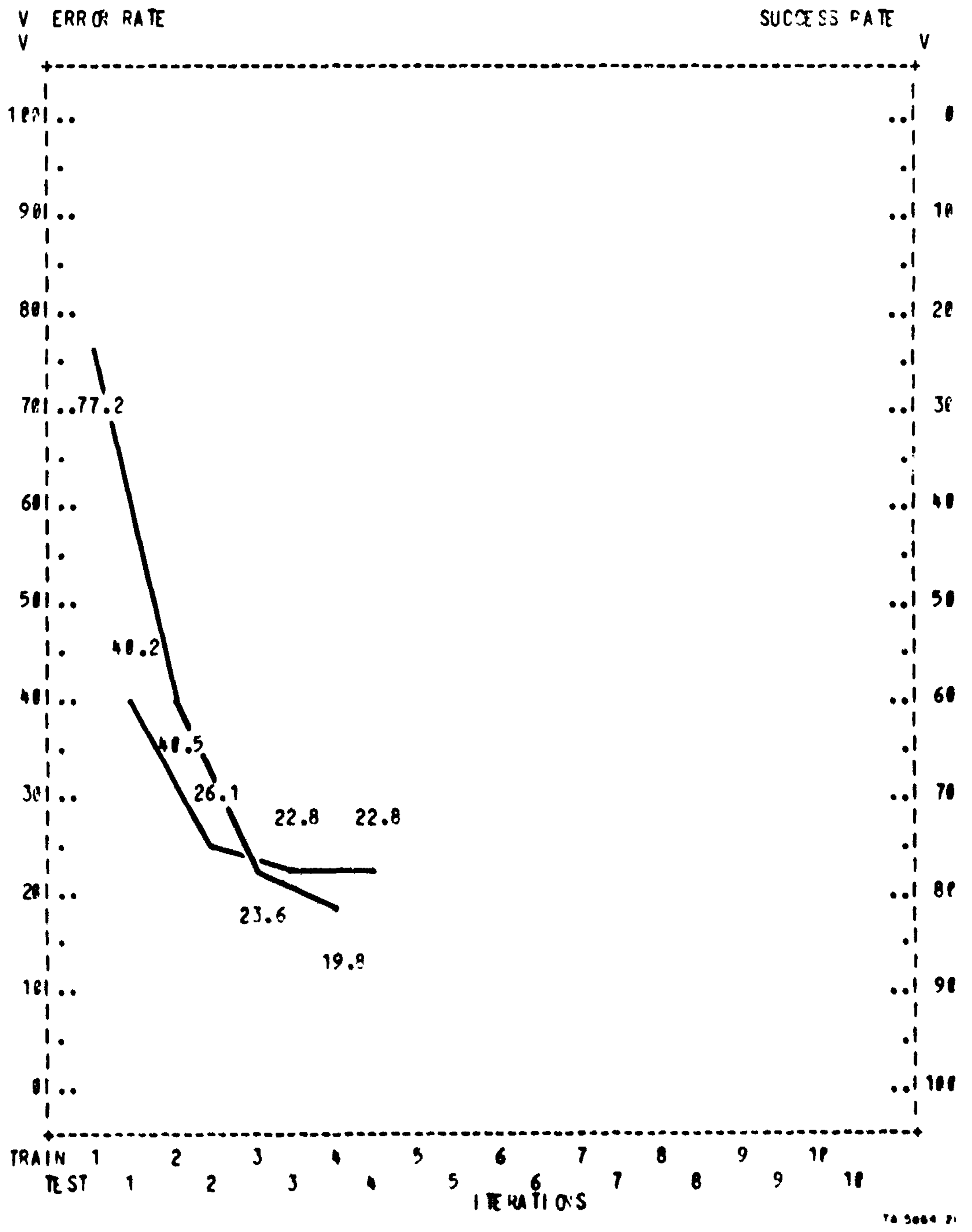


FIG. 2 LEARNING CURVES FOR TOPO-CALM EXPERIMENT 2

LEARNING CURVES

SRI PROJECT 5864, EXPT NO. TOPO-CALM 4
INTRA-AUTHOR EXPT. TRAIN ON 18 ALPHABETS, TEST ON NOS. 50,51.
THERE ARE 828 TRAINING PATTERNS AND 92 TESTING PATTERNS

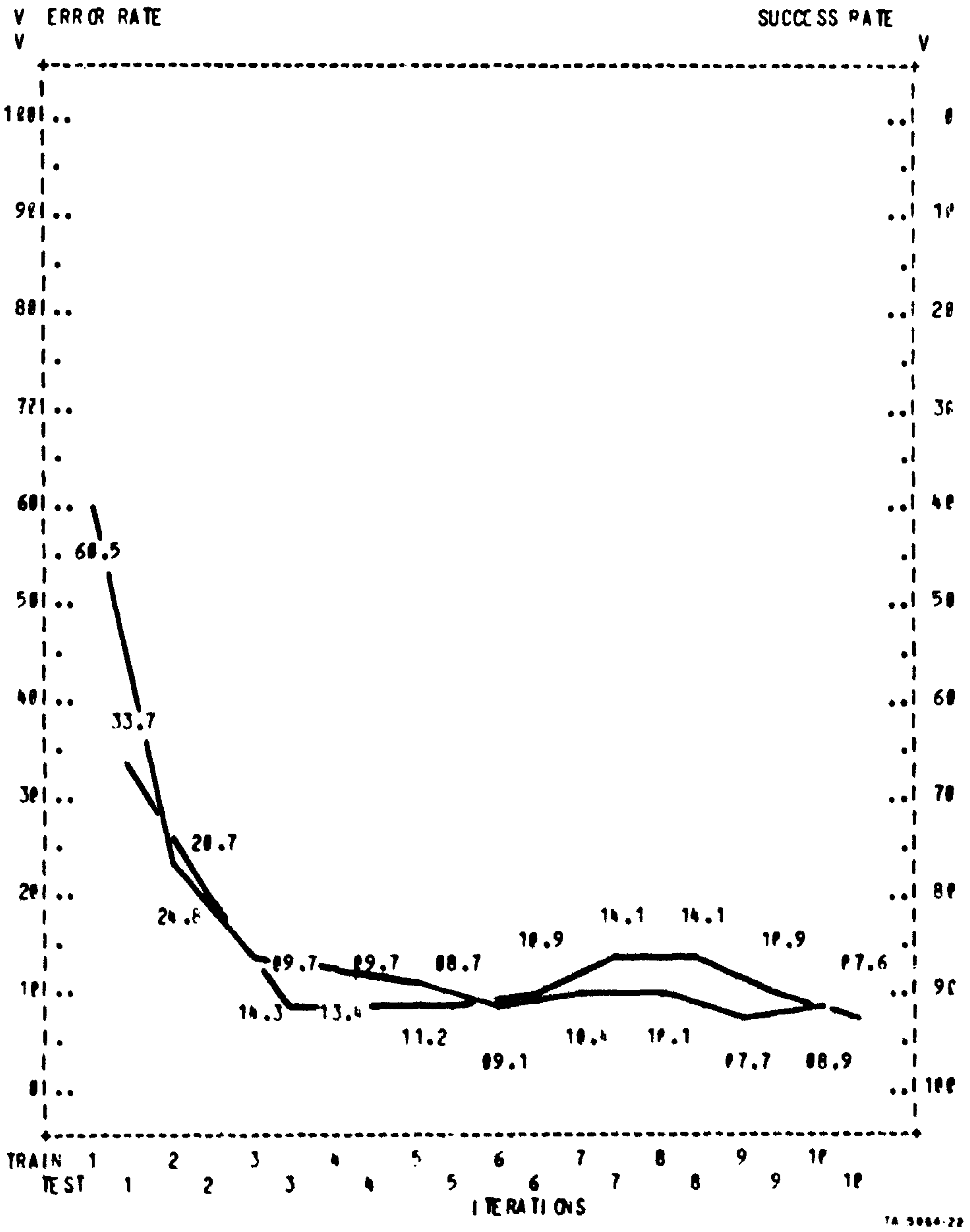


FIG. 3 LEARNING CURVES FOR TOPO-CALM EXPERIMENT 4

LEARNING CURVES
-------- ------

SRI PROJECT 5864, EXPT NO. TOPO-CALM 6
INTRA-AUTHOR EXPT. TRAIN ON 18 ALPHABETS BY ROD, TEST ON NOS. 20,21.
THERE ARE 828 TRAINING PATTERNS AND 92 TESTING PATTERNS

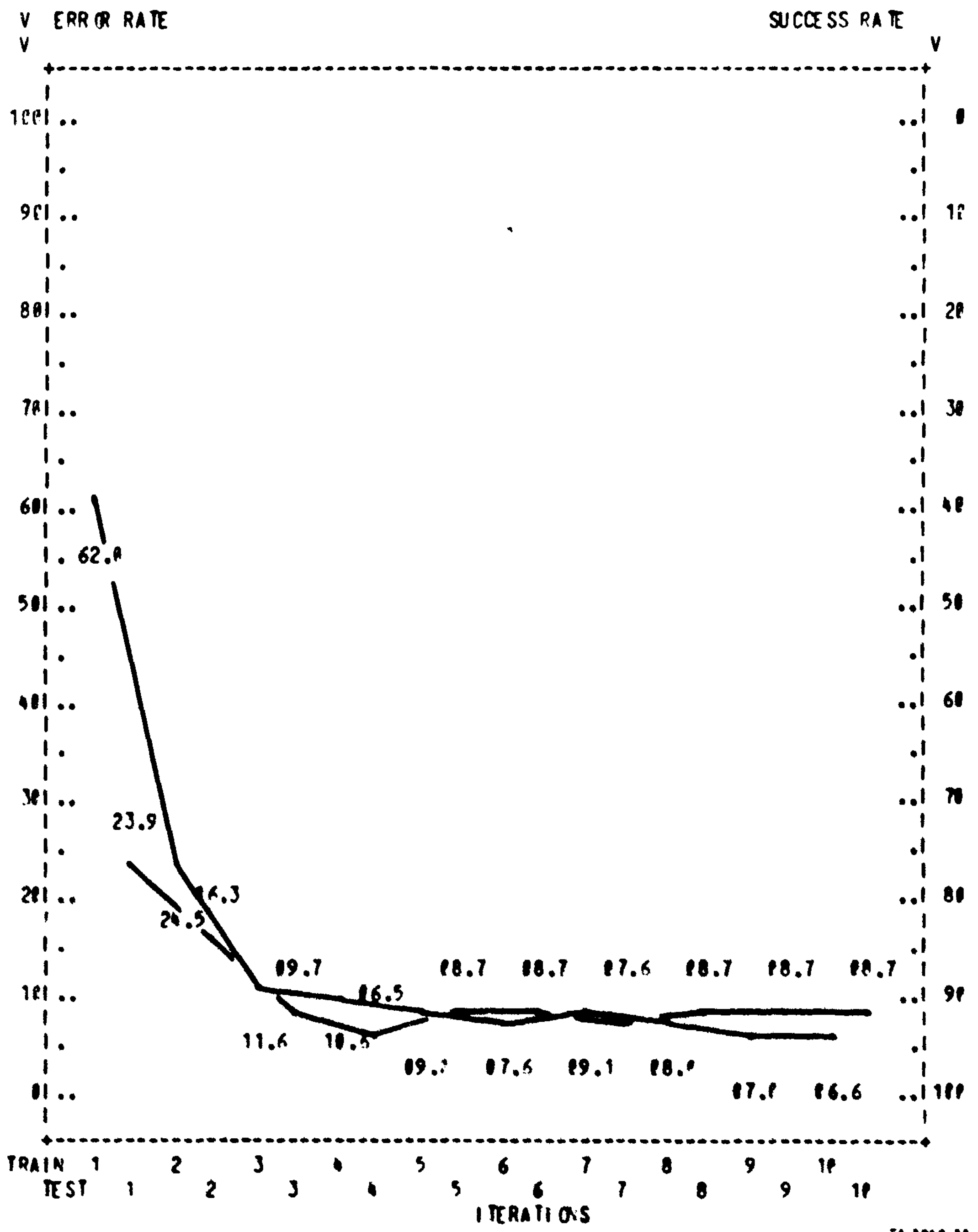


FIG. 4 . EARNING CURVES FOR TOPO-CALM EXPERIMENT 6

4. PREP-CALM Experiment 10

If intra-author testing produces such a marked improvement in performance on patterns preprocessed through TOPO 2 and classified by CALM, should it not do the same for patterns preprocessed by PREP 24? To investigate this question, the file of twenty alphabets by JM was sent through the nine-view, edge-detecting preprocessor simulation, PREP 24A. The resulting feature vectors were classified by CALM, using the same training and test sets as in TOPO-CALM Experiment 4.

By accident, only the centered view of each pattern was used for training and testing. Even so, the training error rate went to zero in six iterations, and the test error rate reached 8 percent. This result compares with test error rates of 25-35 percent for an inter-author experiment with a much larger training set (Experiment 1, in the Second Quarterly Report). This is strong independent evidence in favor of intra-author testing.

5. A Combined TOPO-PREP Experiment

We have previously discussed the possibility of combining the output of two or more classifiers to achieve improved recognition accuracy (Second Quarterly Report, pp. 8-9). With an eye to this possibility, we used the same training and testing pattern sets in PREP-CALM Experiment 10 that we had used in TOPO-CALM Experiment 4. By combining the responses from the classifiers in the two experiments, it was possible to create a new preprocessor-plus-classifier system equivalent to the two individual ones linked together.

We combined the outputs of the classifiers by causing the CALM program to print out, in each case, the maximum and second maximum Dot Product Unit (DPU) sums together with the DPU sum in the desired category. From this information it was possible to determine, when one classifier was in error, whether the other one had a correct vote strong enough to reverse the decision.

The result, for the combined classifier, was three errors in 92 patterns, or a 3 percent test error rate. The test error rates for the two individual classifiers were 7/92 and 5/92, respectively. Although the test sample is small and is dependent on a specific pair of alphabets, the evidence is quite clear that the combined system represents an improvement over the individual ones.

6. Discussion

In this section, we have seen that intra-author testing can lead to a dramatic improvement in test error rate, cutting it from previous lows on the order of 20 percent to less than 10 percent. In addition, the combining of two preprocessor-classifier pairs has cut the error rate by another sizable percentage.

The results of these experiments should be considered somewhat tentative, for two reasons. First, the test samples were statistically small (92 patterns), and limited to two authors. Second, the data were taken from coding sheets in which twenty alphabets were written on successive lines at one sitting. We certainly expect to find more variability among patterns collected from actual coding sheets produced by an author at different times.

At the time of this writing, we are engaged in revising the coding-sheet scanning program to facilitate the gathering of a large amount of data from coding sheets. We are also upgrading the TOPO program, to make it run faster and to improve and expand its set of output features. We anticipate using a body of coding-sheet data from each of several authors to further the investigation of intra-author testing. By using a larger training set, and one taken from coding sheets rather than sheets containing alphabets, we should be able to counteract the negative effect of increased variability in the test patterns produced by an author. The use of actual coding sheets will assure us of a closer match to a realistic operating situation.

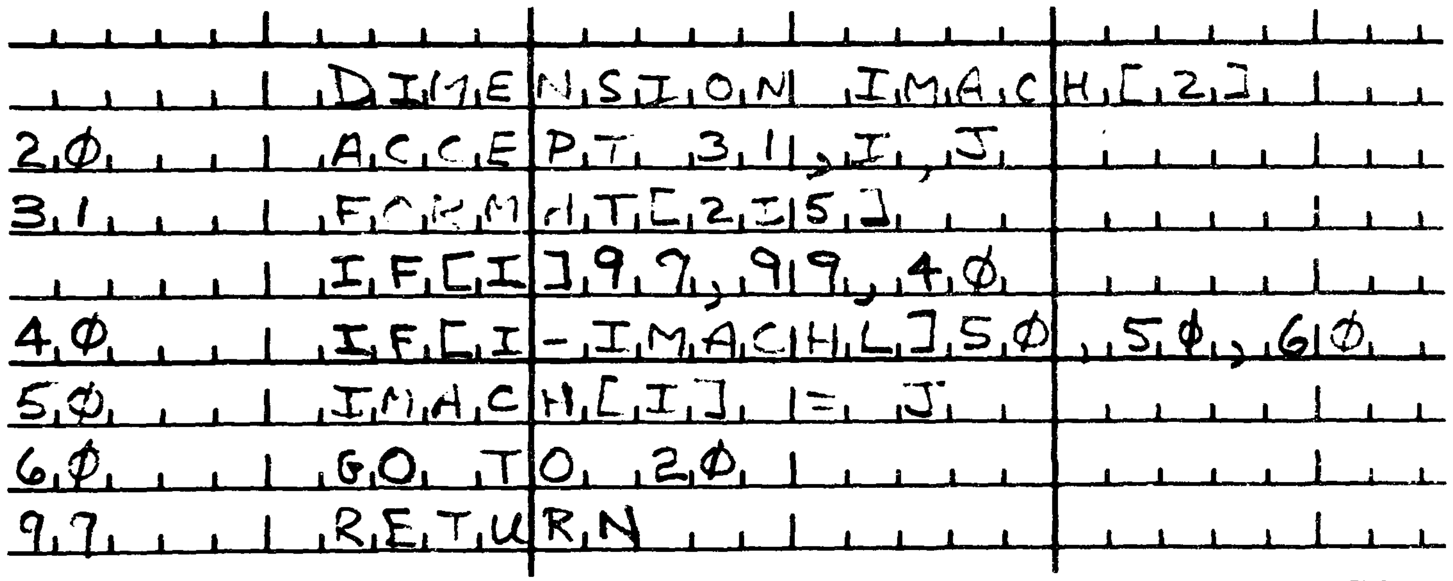
IV AN EXPERIMENT INCLUDING PREPROCESSING, CLASSIFICATION, AND SYNTAX ANALYSIS

An experiment has been performed in which a sample of text from a coding sheet was scanned, preprocessed, classified, and subjected to syntax analysis. The original text fragment on the coding sheet is shown in Fig. 5. The text without guidelines, as the SCAN 2 program sees it, is shown in Fig. 6. The text consisted of 107 non-blank characters. The black-white quantized images of the characters were typed by the computer and assembled in the format of the text to form Fig. 7. Figure 8 shows a few of the quantized images, on an expanded scale. The quantized images are elongated, owing to the different horizontal and vertical spacings of the output typewriter.

The 107 characters were preprocessed by TOPO 2, and the resulting feature vectors were classified by CALM in a test-only run. CALM used the weights that were developed, during TOPO-CALM Experiment 4, by training on 18 alphabets of characters from the same author. The outputs of CALM were the values of the DPU sums for each category, for every character. After scaling the DPU sums to represent approximate confidence values and cutting off low values, the classification output had the following form:

```
D     3    W    V    D    ]    ,    +    U
     17   16   12   12   11   1Ø    9    8

I     I    $    2    [
     34   22   22   2Ø

M     N    7    *    A    H    M    ]    9
     17   15   14   13   1Ø   1Ø    9    9

E     [    =    C
     42   29   28

N     N
    100
```

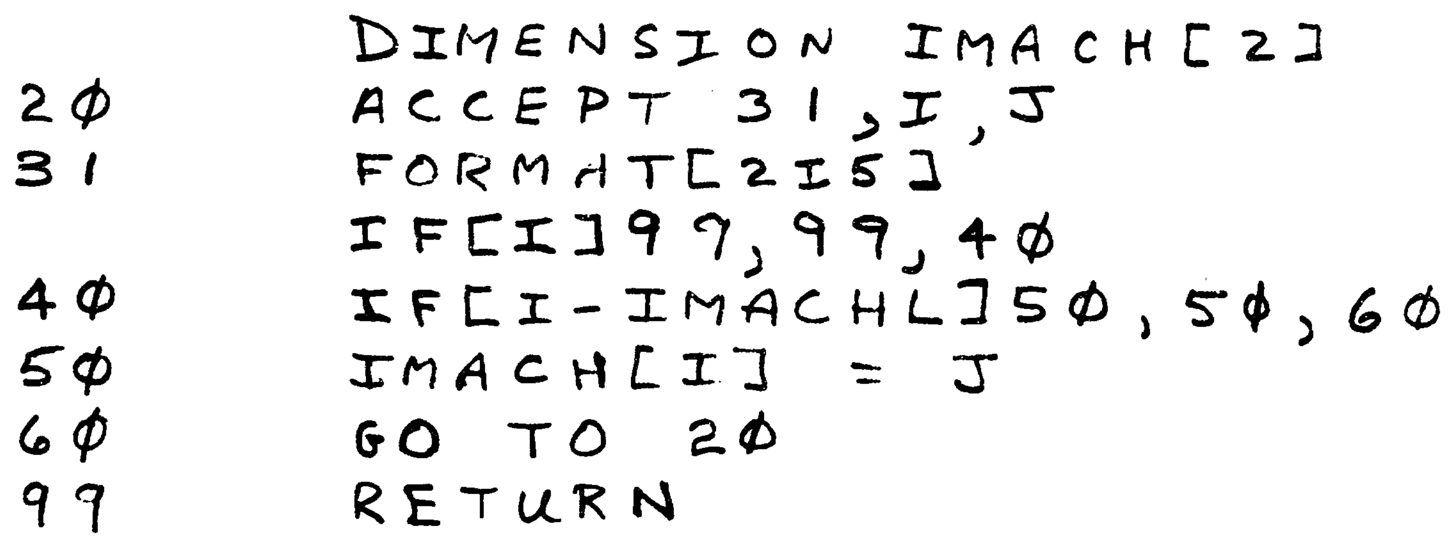
Thus, the first character (D) was classified as a 3 with confidence 17, as a W with confidence 16, and so on. The correct category was in



```
      DIMENSION IMACH[2]
2Φ    ACCEPT 31,I,J
31    FORMAT[2I5]
      IF[I]97,99,4Φ
4Φ    IF[I-IMACHL]5Φ,5Φ,6Φ
5Φ    IMACH[I] = J
6Φ    GO TO 2Φ
97    RETURN
```

TA-5864-24

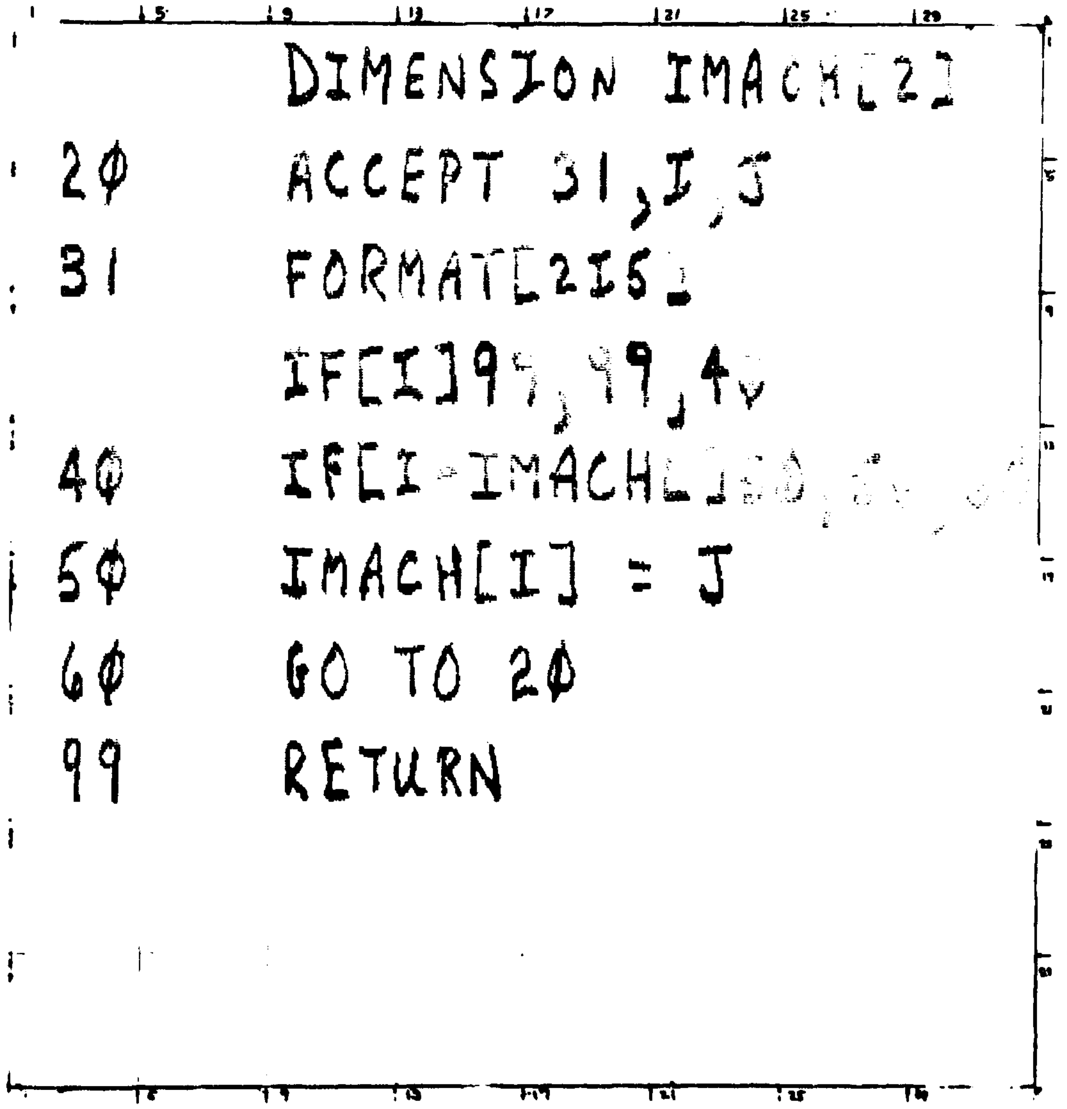
FIG. 5 FORTRAN TEXT ON A CODING SHEET



```
          DIMENSION IMACH[2]
2Ø        ACCEPT 31,I,J
31        FORMAT[2I5]
          IF[I]99,99,4Ø
4Ø        IF[I-IMACHL]5Ø,5Ø,6Ø
5Ø        IMACH[I] = J
6Ø        GO TO 2Ø
99        RETURN
```

TA-5864-25

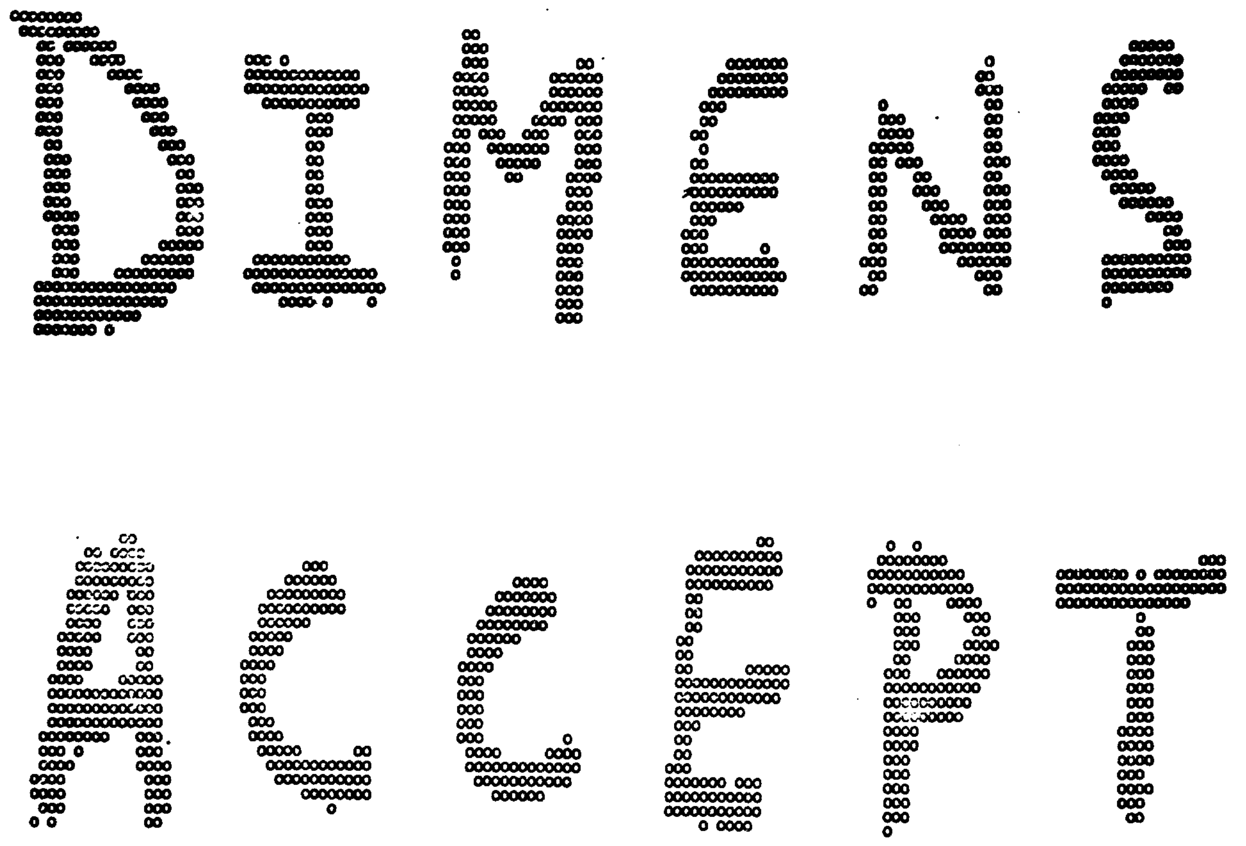
FIG. 6 FORTRAN TEXT ON A CODING SHEET, VIEWED THROUGH A FILTER



```
        DIMENSION IMACH[2]
  2Ø    ACCEPT 31,I,J
  31    FORMAT[2I5]
        IF[I]99,99,4Ø
  4Ø    IF[I-IMACH[I]][illegible]
  5Ø    IMACH[I] = J
  6Ø    GO TO 2Ø
  99    RETURN
```

TA-8864-26

FIG. 7 QUANTIZED IMAGES OF TEXT



TA-5864-27

FIG. 8 QUANTIZED IMAGES OF SEVERAL CHARACTERS

fourth place for this character. For the fourth character (E), the correct category was not even on the confidence list.

By taking the highest-confidence category as the classification for each character, the following text is obtained:

```
      3IN[NSIQN IMACM[2]
20    ACCEPT ]1,I,J
31    FOK*AX[2I5]
      IF[I]79,Z9N40
40    IC[I-IMA[:L]50/50,60
50    IMACH[I]=J
6Q    GQ TD 2B
99    RITKRN
```

This text contains 21 errors, giving an error rate of 21/107, or 20 percent on a character-by-character basis. (This relatively high error rate for intra-author test reflects the differences between the alphabet text that was used for training and the coding-sheet text being tested.)

The confidence lists for the characters were then given as input to the syntax-analysis program, one FORTRAN statement at a time. The syntax-analysis program has been described in this report and in the two preceding Quarterly Reports. Making use of the alternative classifications presented for each character, the syntax analyzer attempts to match the text with a FORTRAN statement type and to resolve classification errors which result in syntactically impossible text.

In this case, the syntax analyzer returned the following text:

```
      DIMENSION IMACM[2]
20    ACCEPT 31,I,J
31    FORMAT[2I5]
      IF[I]79,99,40
40    IF[I-IMACHL]50,50,60
50    IMACH[I]=J
60    GO TO 20
99    RETURN
```

In the cleaned-up text, only two characters are in error. The variable IMACH appears as IMACM in the first line, and the statement number 99 appears as 79 in the fourth line. The classification error rate, on a per-character basis, has been reduced by a factor of ten.

The two errors that remain in the text are, of course, syntactically correct on a statement-by-statement basis, but they are susceptible to analysis in the context of a complete program. The statement number 79 cannot be correct within the fragment shown, because there is no statement labeled 79. Semantic analysis would show that the correct value is very probably 99. This analysis would take account of the occurrences of 99 elsewhere in the fragment, as well as the fact that "9" was the second choice for the misclassified character. Similarly, global and semantic analysis would indicate that IMACM on line 1 and IMACH on line 6 were probably occurrences of the same variable name (in fact a global syntactic error would result otherwise), although it would not be proper to choose one of the two forms as correct on the basis of the fragment shown. Thus, we see examples of both a correctable error and one that is only detectable and should be flagged for a human proofreader. We are currently developing programs for semantic and global analysis of this type.

The experiment described here is the only one of its type that we have performed to date. We plan to do more, in conjunction with the forthcoming additional scanning and intra-author classification of coding sheets. Despite its limited scope, this experiment provides a good example of the type of recognition-rate improvement that we are attempting to achieve with syntactic and semantic analysis.

V STATUS OF THE SYNTAX ANALYSIS PROGRAM

A. Introduction

The most important single problem facing the syntax analysis effort is the resolution of algebraic expressions.* The problem occupies a central position in the work for at least two reasons. First, algebraic expressions form portions of many of the most important FORTRAN statement types, e.g., the arithmetic assignment statement, IF statement, computed GO TO statement, and many input-output statements. Secondly, although the rules for forming algebraic expressions are simple (just the ordinary rules of algebra), the expressions themselves can be of arbitrary complexity. Thus, the development of a program which could successfully handle expressions is not only essential from a practical point of view, but would represent a significant increase in the sophistication of our techniques.

During the past quarter, we have developed a number of approaches to the problem of resolving expressions. In the remainder of this chapter we first point out the inadequacy of a straightforward approach, and then detail the alternatives which are under investigation.

B. Dynamic Programming Approach

The most obvious way to resolve an algebraic expression would be to use dynamic programming in conjunction with the predicate function EXPP. Conceptually, this would insure finding the string of symbols with the highest total confidence of any string that was a legal FORTRAN expression. A little calculation, however, indicates that this method soon faces the prospect of a "combinatorial explosion." Suppose, for example, that a string is 10 characters long, and suppose that the classifier has provided 4 alternatives for each character. Even for this relatively modest example the dynamic programming routine would have to be prepared to generate over one million alternate strings of

*For background, see the two preceding Quarterly Reports.

symbols. Since it requires approximately one computer word merely to specify a single string, it would take a million words of memory just to list the alternatives. Even assuming the availability of such a memory, the computation time would run to at least several hours. Evidently, then, while the straightforward dynamic programming approach is both the conceptually simplest technique and the decision-theoretic optimum one, it cannot be applied directly to the problem of resolving algebraic expressions.

In view of these considerations, the following four alternative techniques have been proposed:

(1) Compression of Characters

(2) Heuristic Partitions

(3) Improvements in Dynamic Programming

(4) Table of Variables.

It is likely that the final method used will incorporate several of these techniques.

C. Compression of Characters

Two independent observations may be made that suggest a way to reduce the combinatorial explosion. First, if the string is n characters long and if there are h choices for each character, then there are h^n possible strings. Thus, complexity is very sensitive to the number of alternatives provided by the classifier for each character. The second observation concerns the FORTRAN syntax. In an isolated algebraic expression, every letter is syntactically equivalent to every other letter, and every digit is syntactically equivalent to every other digit. We may take advantage of this fact by compressing all alternate letters for a single character into a single letter, and do the same for all digits. A simple example will make the method clear. For clarity, we defer for the moment the question of how to combine the confidences associated with the compressed alternatives.

Suppose the original P-list (list of all alternatives for all characters, together with the confidence of each) is:

```
(((I 50) (J 30) (LB 10))
 ((PL 60) (4 20) (H 10) (K 10))
 ((DS 50) (S 30) (B 10) (R 10)))    .
```

This P-list might have come from the original expression "I + S." Ignoring the confidences, we can form the A-list as:

```
((I J LB)
 (PL 4 H K)
 (DS S B R))    .
```

We now compress this list by grouping all occurrences of letters in each sublist into a single letter, say "X," and all occurrences of digits into a single digit, say "1." We leave the special characters (PL,MI,DS, etc., meaning +, -, $, etc.) unchanged, since they each have their own syntactic meaning. The compressed list would then be:

```
((X LB)
(PL 1 X)
(DS X))    .
```

Whereas the original list provided 48 possible distinct strings, the compressed list provides only 12. This compressed list (with the confidences inserted) would be the input to the dynamic programming routine. The output of the routine would be the expression "X + X." At this point the expression would have to be reconstructed, by inserting for each "X" and each "1" an appropriate choice. The simplest way to do this is to use the highest confidence letter or number that the classifier originally provided for each particular character. Reconstructing characters in this fashion is in fact theoretically optimum, if no semantics are employed, i.e., if the global structure of the entire FORTRAN program is not used. In our example, "X + X" would become "I + S."

Returning to the question of combining confidences during compression, we note that the solution hinges upon how we use confidences to represent probabilities. In the Appendix, we show that a reasonable combination of confidences, under the assumption of logarithmic scaling, is to set the confidence of the compressed character equal to the maximum confidence of any of its constituents. In our example, the final result of compressing would be the list:

```
(((X 50) (LB 10))
 ((PL 60) (1 20) (X 10))
 ((DS 50) (X 30)))     .
```

LISP programs for compressing a P-list and reconstructing a legal expression from the output of the dynamic programming routine have been written and debugged. Our experience with them to date indicates that, while the saving in time and storage is considerable, the method is not sufficiently powerful by itself to solve the problem of combinatorial complexity of algebraic expressions.

D. Heuristic Partitioning

Another approach to containing the combinatorial explosion depends upon the fact that an algebraic expression is composed of a sequence of elements connected by one of the five arithmetic infix operations. We can partition an expression into a sequence of elements by using any or all of the five operators (+, -, *, /, **) as delimiters. Since the elements between operators are often themselves expressions, the essence of this approach is to try to break up one long expression into a sequence of shorter ones, resolve each of the shorter ones, and then string them together to form the original expression. In practice, such an approach might be implemented in the following manner.

(F r simplicity, we assume that only the operator "+" will be used as a delimiter.) Given a P-list, we do the following:

(1) Mark all L-lists containing "+" among the alternatives. (An L-list is a list of the alternatives and confidences that the classifier provides for a single original character, i.e., a top-level element of a P-list.)

(2) Select some subset of the marked L-lists as being delimiters.

(3) The delimiters partition the original P-list into segments. Try to extract an algebraic expression from each segment.

(4) If successful, string the expressions found in Step 3 together with "+" and stop. If not, select some other subset of the marked L-lists as being delimiters, and proceed to Step 3.

In the example of the preceding section, there is only one possible "+." The original P-list would then be broken up into the two segments ((I 50) (J 30) (LB 10)) and ((DS 50) (S 30) (B 10) (R 10)). Each segment would be resolved into the simple expressions "I" and "S" respectively, and the final expression "I + S" would be formed.

We have written and debugged two LISP programs that implement the above algorithm. The first version chooses the delimiters (in Step 2 of the algorithm) according to the rule "have as many delimiters as possible." This tends to break the original P-list into many small pieces, so that the computational cost of resolving each piece is relatively low. The second version uses dynamic programming to select delimiters (also in Step 2), and is more sophisticated in its methods for avoiding repetitious computation. The first version has the advantage of being computationally feasible, but its inattention to the confidences of the operators used as delimiters means that it often returns legal expressions of low total confidence. The second method avoids this pitfall, but is more likely to encounter situations where the length of a segment is still too long to be handled by available techniques. Both versions suffer from the fact that it is perfectly possible to find a legal expression with "+" signs in which the segments delimited are not legal expressions. For example, if VAR is an array variable, then VAR(I + S) is a legal expression, but both "VAR(I" and "S)" are not legal.

E. Improvements in Dynamic Programming

We may recall (see the Fourth Quarterly Report, p. 40) that the first part of the dynamic programming routine builds a graph of possible strings of characters. Once the graph is built, then the second part of the routine peels off one alternative string at a time, in order of decreasing confidence, and hands the string over to another program to check legality. (In this case, the other program would be the function EXPP, which checks for legal expressions.) It is quite clear that we would never want to peel off strings near the bottom of the graph, since the overall confidence of such a string would be low. Therefore, the bottom portion of the graph can be neglected. The first implementation of the dynamic programming routine nevertheless built the entire graph. To remedy this, two implementations of the building portion of the dynamic programming routine have recently been completed that are sophisticated enough to avoid building the lower portion of the graph. Although there is no doubt that these improved versions will vastly reduce the memory and time requirements of dynamic programming, we have not yet gained enough experience with them to specify quantitatively the maximum complexity they can handle.

F. Table of Variables

The final method under consideration is the only one of the four to exploit the semantics of a particular program directly, in addition to using the general constraints of the syntax of FORTRAN. We observe first that names of variables can be up to eight characters in length, and thus, typically constitute the bulk of an expression. Second, we observe that many variables appear more than one time in a typical program, often appearing first in the COMMON or DIMENSION statements. If the variables in a program could be identified initially, then one might scan through expressions and isolate at least some of the variable names, replace the corresponding segments of the P-list with simple symbols, and thus reduce the computational burden.

At present, this approach is the least explored of the four techniques presented, and an evaluation of its utility cannot be made until it is more fully developed. We expect to explore this approach as a natural corollary to the general semantic analysis of program texts.

APPENDIX

In this Appendix we show that, with logarithmic scaling, the confidence of a compressed character is approximately equal to the maximum of the confidence of its constituents. Suppose the characters $a_1, \ldots,$ a_n are compressed into a single character a. Let c_i be the confidence and p_i be the probability of a_i, and let c and p be the confidence and probability of a. Then

$$c_i = \ln p_i \quad ,$$

$$p_i = e^{c_i} \quad ,$$

$$c = \ln p \quad ,$$

$$p = e^{c} \quad ,$$

by the definition of our confidences. We can then write

$$p = \Pr\{a\} = \Pr\{a_1 \cup a_2 \cup \ldots \cup a_n\}$$

$$= \sum_{i=1}^{n} \Pr\{a_i\} = \sum_{i=1}^{n} p_i = \sum_{i=1}^{n} e^{c_i} \quad ,$$

since the events a_i are mutually disjoint. But $c = \ln p$, so

$$c = \ln \left[\sum_{i=1}^{n} e^{c_i} \right] \quad .$$

If we define $c_{max} = \max_i c_i$, we have

$$\ln\left[e^{c_{max}}\right] \le c \le \ln\left[n\, e^{c_{max}}\right] \quad ,$$

or

$$c_{max} \le c \le c_{max} + \ln(n) \quad .$$

Thus a conservative approximation is to set $c = c_{max}$.

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Stanford Research Institute 333 Ravenswood Avenue Menlo Park, California 94025	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP N/A

3. REPORT TITLE

GRAPHICAL-DATA-PROCESSING RESEARCH STUDY AND EXPERIMENTAL INVESTIGATION

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Report No. 28, Sixth Quarterly Report--1 June 1967 to 31 August 1967

5. AUTHOR(S) *(First name, middle initial, last name)*

Peter E. Hart; John H. Munson

6. REPORT DATE September 1967	7a. TOTAL NO. OF PAGES 42	7b. NO. OF REFS 0
8a. CONTRACT OR GRANT NO. DA-28-043-AMC-01901(E) b. PROJECT NO.	9a. ORIGINATOR'S REPORT NUMBER(S) SRI 5864 Sixth Quarterly Report	
c. d.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)* ECOM-01901-28	

10. DISTRIBUTION STATEMENT

Distribution of the document is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY U.S. Army Electronics Command Fort Monmouth, New Jersey 07703

13. ABSTRACT

This report describes the continuing development of scanning, preprocessing, character-classification, and context-analysis techniques for hand-printed text, such as computer coding sheets in the FORTRAN language.

The performance of topological feature extraction, combined with character classification by a learning machine, is described, and compared with the performance of other combinations. By performing intra-author testing (gathering the training data and test data from the same author), we have achieved a dramatic reduction in test error rate, to less than 10 percent on a limited sample.

We describe an experiment in which a fragment of FORTRAN text is scanned, preprocessed, classified character by character, and subjected to context analysis to greatly reduce the recognition-error rate. Finally, we discuss advances in the method of analyzing arithmetic expressions, a key aspect of the context analysis.

DD FORM 1 NOV 65 1473 (PAGE 1)
S/N 0101-807-6801

UNCLASSIFIED
Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Pattern Recognition						
Adaptive Systems						
Learning Machines						
Feature Extraction						
Classification						
Context Analysis						
FORTRAN Syntax Analysis						